Reduction with Complete Unrolling:

In this video, we are going to show you how to completely unroll the iterations of the for loop in our reduction implementation. In previous videos, we saw how to remove warp divergence faced by first warp by unrolling last six iterations in the for loop. If our thread block size is more than 128 threads, let say we have thread block with 1024 threads, then in this for loop, starting offset size will be 512, then in each iteration, we are going to reduce that offset value by half until offset become 64. So let's try to visualize each of these iterations now. In the first iteration, our offset value is 512, and we will sum up first 512 elements with corresponding second 512 elements, and store the results back to first 512 indices in our array. After this step, each element in first 512 indices has summation of 2 elements. Summation of both element with same index in original array and element with 512 offset. For example after this step first element will have the summation of first element and 512 element in original array and so on. In the second iteration our offset size is 256 and first 256 elements will sum up with elements which are at 256 offset away, so after this iteration, first element will have the summation of 1st, 512th, 256th, and 768th elements in the original array. In the 3rd iteration, first 128 elements at this stage sum up with elements with 128 offset away from those. And in the 4th iteration, first 64 element at this stage sum up with the element 64 offset away from those and after this iteration, we are done with our for loop. If we can manually do what this for loop is doing for us, then we can reduce loop maintenance instruction overhead. , So let's try to completely unroll "for" loop from our reduction kernel implementation. This kernel is going to be very similar to our warp unrolling implementation, so let me copy that kernel and paste it here, and change its name to reduction complete unroll. ok, our purpose is to replace this "for" loop. So let me remove that "for" loop from our code first. Now if our thread block size is 1024, then in the first iteration we have to sum up elements with 512 offset values. So let me add that condition check here. If our thread block size is 1024, then first 512 threads should perform the summation. Next if we consider data blocks with 512 elements, then we can sum up elements with 256 offset. We can do the same operation considering offset value 128 and, 64 as well. So in this way, we can completely unroll the for loop. Now after this, there is no for loop in our implementation. Let me walk you through this implementation, assuming 512 threads as the thread block size now. If the thread block size is 512, then the first condition check is going to fail. So elements summation with 512 as offset, is not going to perform here. But all other condition checks will be true for thread block with 512 threads, , So it will perform summation with offset 256, 128, and 64 respectively before moving on to the warp unrolling section. Let's run this example and check the validity of it. Ok, here you can see, CPU and GPU results are same, which means this implementation also yield the correct results.